Remote Agent Access

This guide is for integrators who run an OmniLink agent on a different machine than the one they use to drive it — for example, an agent that lives on a robot's onboard computer and is controlled from an operator's laptop, an industrial PC in a sealed enclosure, or an edge server in a remote facility. It covers what the agent actually exposes over the network, why a hosted HTTPS page appears to only reach `localhost`, and every recommended setup for both local-network and internet access — with commands, configuration, security guidance, and troubleshooting.

---

1. The Mental Model

An OmniLink integration always involves at least three actors. Knowing which one is which makes every setup choice in this guide obvious.

Actor	Where it runs	What it does
Web UI	The operator's browser, either on the hosted page at `https://www.omnilink-agents.com` or a self-hosted copy.	Chat surface and Connection panel. Sends commands and reads back state.
Agent process	The machine that controls the physical thing — robot computer, edge server, factory PLC bridge, etc.	Runs your `OmniLinkEngine` + `OmniLinkHTTPBridge` (or a `ToolRunner`) and any tool implementations.
Hardware bridge	Same machine as the agent, usually loopback.	Translates agent commands into hardware actions (Webots, ROS, GPIO, OPC UA, vendor SDK).

The networking question this document answers is exactly one thing: **how does the browser reach the agent process when the two are not on the same machine?**

```text
Operator's browser                    Agent host (e.g. robot computer)
──────────────────                    ──────────────────────────────────
 OmniLink Web UI       ◀──HTTP/S──▶     OmniLinkHTTPBridge      :5000
     (Connection                       Tool-callback server    :5xxxx
      panel URL)                       Hardware bridge         :8765 (loopback)
                                       Physical device         (USB/CAN/GPIO/…)
```

2. What the Agent Exposes

A typical OmniLink agent (Axis, Haven, Jasmine, or your own) opens two small HTTP services on the host machine. You need to know what they are because **both** must be reachable by the browser for a working integration — forgetting the tool-callback port is the most common cause of "chat works, tools don't".

2.1 Automation bridge

Property	Value
Class	`OmniLinkHTTPBridge` (in `omnilink-lib`)
Default port	`5000`
Default bind	`0.0.0.0` (every interface)
Endpoints	`POST /command`, `POST /inline-code`, `GET /context`, `GET /feedback`
Auth	None built-in. CORS is set to `*`.
Used by	The web UI's Connection panel.

This is the "command channel" — the UI POSTs commands here and polls the feedback/context endpoints to render results.

2.2 Tool-callback server

Property	Value
Started by	The agent itself (Axis, Haven, your `ToolRunner` subclass).
Default port	Per-agent. Axis: `AXIS_PORT` (default `51516`). The bundled `ToolRunner` picks a free port.
Default bind	**Axis-style robot agents:** `0.0.0.0`. **Bundled `ToolRunner`:** `127.0.0.1` only.
Endpoints	`POST /tool` (run a tool), `GET /activity` (recent activity log)
HTTPS support	Optional via `use_https=True` in `ToolRunner` — emits a self-signed cert cached under `~/.omnilink/ssl/`.
Private-Network header	Always sends `Access-Control-Allow-Private-Network: true` for Chrome's PNA preflights.
Used by	The web UI when the AI emits a structured tool call.

The bundled `ToolRunner` binds to loopback on purpose — on a single-user workstation, anything on the LAN that could reach it could invoke the agent's tools. Robot-style agents like Axis bind to `0.0.0.0` because the cloud UI is, by definition, not on the same machine.

2.3 Anything else?

Your hardware bridge (Webots, ROS bridge, vendor SDK) may also speak HTTP, but the browser never talks to it directly — the agent process is the only client that connects to it, so it stays on loopback.

---

3. Why “Localhost Only” Appears

The hosted web UI is served from `https://www.omnilink-agents.com`. Modern browsers enforce a **mixed-content** rule: an HTTPS page is not allowed to make plain HTTP requests to arbitrary hosts. If you point the Connection panel at `http://192.168.1.50:5000`, the browser silently blocks the request — you get a CORS-shaped error in DevTools, but the underlying cause is mixed content.

There is exactly one exception. Browsers treat the following as **potentially trustworthy origins** and allow HTTPS pages to fetch from them:

- `http://localhost` and any DNS name that resolves to a loopback address
- `http://127.0.0.1` (and the entire `127.0.0.0/8` block)
- `http://[::1]`

That’s why an SSH port-forward “works” — once you tunnel the robot’s port to `localhost:5000` on the operator laptop, the URL looks loopback to the browser, and the mixed-content rule lets it through.

The rule of thumb is therefore very simple:

> **The agent’s URL, as the browser sees it, must be `localhost` / `127.0.0.1` / `[::1]` or an `https://` URL with a trusted certificate.** Everything in this guide is a way to satisfy that.

Chrome adds a second wrinkle: **Private Network Access (PNA).** A page on a public origin that fetches a private IP (RFC 1918) requires the target to acknowledge the request with `Access-Control-Allow-Private-Network: true` on the preflight. The OmniLink tool servers already emit this header, so you don’t need to think about it; just be aware that DevTools

may surface a distinct error for it if you forget the same header on a reverse proxy in front of the bridge.

---

4. LAN Setups (Operator and Agent on the Same Network)

All four options below produce a working integration on a local network. They differ in how much setup is needed on each side and how production-ready they are.

4.1 Option A — SSH / `socat` port forward to `localhost`

Forward the bridge port (and the tool-callback port) from the operator laptop to the loopback address on the robot host:

```bash
# From the operator laptop (Linux / macOS / WSL)
ssh -L 5000:127.0.0.1:5000  user@robot.local
ssh -L 51516:127.0.0.1:51516 user@robot.local

# Or in one session, forwarding both:
ssh -N \
    -L 5000:127.0.0.1:5000 \
    -L 51516:127.0.0.1:51516 \
    user@robot.local
```

On Windows (PowerShell with OpenSSH built in):

```powershell
ssh -N `
    -L 5000:127.0.0.1:5000 `
    -L 51516:127.0.0.1:51516 `
    user@robot.local
```

If you can't use SSH (e.g., a Windows robot host that only exposes WinRM), `socat` works on the operator side too:

```bash
socat TCP-LISTEN:5000,fork,reuseaddr TCP:192.168.1.50:5000 &
socat TCP-LISTEN:51516,fork,reuseaddr TCP:192.168.1.50:51516 &
```

In the Connection panel, use `http://localhost:5000`. The agent profile's `toolCallbackUrl` already points at `http://127.0.0.1:51516/tool`, which the browser resolves locally and the tunnel carries to the robot.

- **Pros:** Zero changes on the robot. Encrypted hop. Works through most NATs and corporate firewalls.
- **Cons:** The tunnel only exists while the SSH/socat session is up. Every port you care about needs its own `-L`. Not a long-running answer.

4.2 Option B — Self-host the OmniLink web UI

Run the web UI yourself, on the operator laptop or any host on the LAN, served over plain HTTP. Now both the UI page and the agent are `http://…` origins, so the mixed-content rule does not apply.

```bash
git clone https://github.com/omnilink-tech/olink
cd olink
npm install
npm run dev          # serves the UI on http://localhost:5173
# or a static preview of the production build:
npm run build && npm run preview
```

In the Connection panel, point at `http://192.168.1.50:5000` directly. The tool-callback URL likewise needs to use the robot's LAN IP — for Axis, change the default by setting `AXIS_PORT` to a known port and patch your agent so it pushes `http://<robot-ip>:<port>/tool` into the profile (the same code path that already sets `toolCallbackUrl`).

- **Pros:** Direct, no tunnel, no certificates. Full control of UI versions.
- **Cons:** You're responsible for keeping the UI up to date. Auth happens at the UI layer (no Supabase session by default if you're self-hosting without the env vars).

4.3 Option C — Put HTTPS in front of the agent (Caddy / nginx)

Run a small reverse proxy on the robot computer that terminates TLS in front of the bridge. The hosted UI can then talk to `https://robot.local` directly.

Minimal `Caddyfile`:

```
robot.local {
    tls internal
    encode gzip

    handle /command {
        reverse_proxy 127.0.0.1:5000
    }
    handle /feedback {
        reverse_proxy 127.0.0.1:5000
    }
    handle /context {
        reverse_proxy 127.0.0.1:5000
    }
    handle /inline-code {
        reverse_proxy 127.0.0.1:5000
    }
    handle /tool {
        reverse_proxy 127.0.0.1:51516
    }

    header {
        Access-Control-Allow-Origin "*"
        Access-Control-Allow-Methods "GET, POST, OPTIONS"
        Access-Control-Allow-Headers "Content-Type, Authorization"
        Access-Control-Allow-Private-Network "true"
    }
}
```

`tls internal` produces a Caddy-managed local CA. The first time you visit `https://robot.local` from a new operator laptop, you need to import Caddy's root certificate into the OS trust store, otherwise the browser refuses the connection.

Import the Caddy root on Windows (PowerShell):

```powershell
# On the robot host, copy the root cert from Caddy's data dir:
#   %LocalAppData%\Caddy\pki\authorities\local\root.crt
# Then on the operator laptop:
Import-Certificate -FilePath .\root.crt -CertStoreLocation Cert:\LocalMachine\Roo
```

- **Pros:** No tunnel, no port forward, works straight from the hosted UI. Real `https://` URL.
- **Cons:** One-time certificate trust step on every operator device.

4.4 Option D — mDNS (Bonjour / Avahi) + Caddy for a friendly name

If you don't want to type IP addresses, layer mDNS on top of Option C. Most consumer routers and developer laptops already resolve `*.local` names via mDNS. Linux robots need Avahi installed; Windows hosts publish their hostname via Bonjour if iTunes/Print Services for Windows is installed, or you can run `avahi-daemon` in WSL.

```bash
# Linux robot host
sudo apt install avahi-daemon
sudo hostnamectl set-hostname robot
# Now reachable as http://robot.local from the same broadcast domain.
```

---

5. Remote Setups (Operator Anywhere on the Internet)

Remote access means the operator's laptop and the agent host are not on the same network. You need either a tunnel that pulls the agent's ports up to a public URL, or a private network that pretends they are.

5.1 Option E — HTTPS tunnel (Cloudflare Tunnel, ngrok, Tailscale Funnel)

Run a tunnel daemon on the robot host. The daemon publishes a public HTTPS URL that forwards back to a local port.

```bash
# Cloudflare Tunnel (free, no signup needed for quick mode)
cloudflared tunnel --url http://localhost:5000
# → prints e.g. https://random-words-1234.trycloudflare.com

# ngrok (free tier, URL rotates on restart)
ngrok http 5000
# → prints e.g. https://abc123.ngrok-free.app
```

Drop the resulting URL into the Connection panel. **You also need a second tunnel for the tool-callback port**, and you must update the agent so it pushes *that* public URL into its profile as `toolCallbackUrl`:

```bash
cloudflared tunnel --url http://localhost:51516
# → https://another-random-1234.trycloudflare.com
# Then, before starting Axis, set:
export AXIS_TOOL_CALLBACK_URL="https://another-random-1234.trycloudflare.com/tool"
python agents/axis/axis_agent.py
```

For long-running setups, use named Cloudflare tunnels with a stable hostname (`cloudflared tunnel create` + a DNS record on your zone) so the URLs don't rotate on restart.

- **Pros:** Works from anywhere with internet. No firewall changes. HTTPS with a real, trusted certificate. Easy demos.
- **Cons:** Anyone with the URL can reach the agent — put auth in front of it (see Security). Free tunnel URLs rotate on restart.

5.2 Option F — Mesh VPN (Tailscale, ZeroTier, WireGuard) *(Recommended)*

Install Tailscale (or equivalent) on the robot host and on every operator device. Each device gets a stable private address that is reachable from anywhere either side has internet — nothing is exposed to the public internet, ACLs are central, and the addresses survive moving between networks.

```bash
# On both machines
curl -fsSL https://tailscale.com/install.sh | sh
sudo tailscale up
# Each machine prints its tailnet IP (e.g. 100.x.y.z) and MagicDNS name.
```

Once on a tailnet, combine with one of the earlier options:

- **Tailscale + Option B** (self-host UI on the laptop) — point at `http://<robot-tailnet-ip>:5000`. Cleanest experience, no certs needed because both sides are HTTP and the transit is already encrypted by Tailscale.
- **Tailscale + Option C** (Caddy with a real cert) — use Tailscale's HTTPS feature (`tailscale cert`) so Caddy can serve a Let's-Encrypt-issued cert for `robot.your-tailnet.ts.net`. The hosted UI then talks to the robot over a real `https://` URL with zero public exposure.

Lock the bridge down to the tailnet interface only:

```bash
# Bind the bridge only to the Tailscale interface
python -c "
from omnilink import OmniLinkEngine, OmniLinkHTTPBridge
engine = OmniLinkEngine()
OmniLinkHTTPBridge(engine, host='100.x.y.z', port=5000).loop_forever()
"
```

And lock down who on the tailnet may reach it via an ACL:

```json
{
  "acls": [
    {
      "action": "accept",
      "src": ["tag:operators"],
      "dst": ["tag:robots:5000", "tag:robots:51516"]
    }
  ],
  "tagOwners": {
    "tag:operators": ["group:engineers"],
    "tag:robots":    ["group:engineers"]
  }
}
```

- **Pros:** Stable addresses, encrypted transit, no public exposure, central ACLs. Survives the robot moving networks (factory, lab, customer site).
- **Cons:** Every operator device needs the VPN client installed and authenticated. Adds one piece of infrastructure to manage.

5.3 Option G — Public DNS + reverse proxy + Let's Encrypt

For a permanently online robot or edge server with a public address, give it a real DNS name and have Caddy fetch a Let's Encrypt certificate automatically.

```
robot.yourcompany.com {
    encode gzip

    @bridge path /command /feedback /context /inline-code
    handle @bridge {
        reverse_proxy 127.0.0.1:5000
    }
    handle /tool {
        reverse_proxy 127.0.0.1:51516
    }

    basic_auth /* {
        operator $2a$14$…hashed-password…
    }
}
```

The Connection panel then talks to `https://robot.yourcompany.com` with a real certificate, the browser is happy, and basic auth gates access.

- **Pros:** Clean URL, real cert, no extra clients on operator devices.
- **Cons:** Requires DNS, a public IP (or port-forward on your router), and a serious auth layer — you're putting a robot on the open internet.

---

6. Security — Read This Before Exposing Anything

The bridge ships with **no authentication** and `CORS: *`. On loopback that's fine — the operating system already gates access. Anywhere else, the bridge must sit behind an auth layer. If you skip this on a remote setup, anyone who guesses or scrapes the URL can issue commands to the agent — and for a robot agent, that includes motion commands.

6.1 Auth options, simplest to strongest

Layer	Where it lives	Good for
Basic auth at the reverse proxy	Caddy (`basic_auth`), nginx (`auth_basic`)	Small teams; quick to set up; one password per operator.
Bearer-token middleware	Tiny FastAPI/Express shim that wraps the bridge.	When you want per-operator tokens you can revoke without touching the proxy.
mTLS	Reverse proxy with `client_auth`.	Per-device client certificates; revocation per device.
Tailscale / VPN ACLs	The VPN controller.	Restrict *which devices on the tailnet* can reach the bridge port at all.
OIDC at the proxy	Caddy + caddy-security, oauth2-proxy.	SSO with Google/Okta/Entra; production deployments.

6.2 Apply the same protection to the tool callback

Easy to forget: the tool-callback server has the exact same exposure profile as the bridge. If you tunnel one, tunnel both. If you put basic auth in front of the bridge, put it in front of the tool callback too — the AI's tool calls are what actually *do* things in the world.

6.3 Bind tightly when in doubt

For LAN-only deployments, override the bridge bind to the specific interface you want exposed rather than leaving it on `0.0.0.0`:

```python
from omnilink import OmniLinkEngine, OmniLinkHTTPBridge

engine = OmniLinkEngine()
# Only listen on the LAN interface, not the WAN/Wi-Fi one:
OmniLinkHTTPBridge(engine, host="192.168.1.50", port=5000).loop_forever()
```

6.4 Operational hygiene

- Rotate the operator's password (or token) when a teammate leaves.
- Log `POST /command` at the reverse proxy with timestamp + auth subject — you want a record of who sent which command if a robot does something unexpected.
- Make sure `stop_robot` (or the equivalent safety command for your agent) is reachable through the same channel as everything else, and ideally also reachable via an out-of-band path (physical e-stop is still the law).

---

7. Choosing a Setup

Your situation	Recommended option
Same room, prototyping	**A** (SSH forward) — works in five minutes, no agent changes.
Office LAN, multiple operators	**B** (self-host the UI) — simplest, no certs.
Production fleet on a corporate network	**C** (Caddy + internal CA) for stable infrastructure, or **F** (Tailscale) if devices roam.
Remote demo or one-off access	**E** (Cloudflare Tunnel) with basic auth in front.
Always-on remote control	**F** (Tailscale) — best ergonomics, real security, survives network changes.
Customer-facing / public product	**G** (real DNS + Let's Encrypt) with full auth, ideally combined with VPN-only operator access.
Air-gapped / no internet	**B** + a LAN-local DNS entry, or see OmniLink Edge for the fully self-contained appliance build.

8. Worked Examples

8.1 Robot in a lab, operator at a desk in the same building

Single operator, one robot, both on the lab Wi-Fi. The robot host is a NUC running Ubuntu with Axis.

```bash
# On the robot host
sudo tailscale up
export AXIS_PORT=51516
export OMNI_KEY="olink_…"
python agents/axis/axis_agent.py

# Caddy on the same host, serving https://nuc.tail9abcd.ts.net
caddy run --config /etc/caddy/Caddyfile
```

On the laptop: install Tailscale, open the hosted UI, set the Connection panel to `https://nuc.tail9abcd.ts.net`. The tool-callback URL the agent advertises should resolve to the same hostname.

8.2 Industrial robot on a factory floor, remote operator

Robot is in a factory; engineering team is remote. Production policy forbids public exposure.

1. Install Tailscale on the robot host with SSO restricted to your engineering Google group.
2. Tag the robot `tag:robots`; tag laptops `tag:operators`.
3. Define an ACL that lets only `tag:operators` reach ports `5000` and `51516` on `tag:robots`.
4. Run Caddy on the robot with `tls internal` for a clean URL on the tailnet.
5. Configure the UI's Connection panel to point at the robot's MagicDNS name.
6. Keep a hardware e-stop on the robot; the bridge is your control plane, not your safety plane.

8.3 Demo for a customer over the open internet

```bash
# On the robot host
```

```
cloudflared tunnel --url http://localhost:5000   # → URL_A
cloudflared tunnel --url http://localhost:51516  # → URL_B

export AXIS_TOOL_CALLBACK_URL="URL_B/tool"
python agents/axis/axis_agent.py
```

Drop URL_A in the Connection panel. Put basic auth on both tunnels via a tiny Caddy in front (or use a Cloudflare Access policy). Take it down when the demo is over — do not leave it up.

8.4 Mobile or embedded robot moving across networks

For a robot that moves between Wi-Fi networks (warehouse aisles, customer sites), Tailscale is effectively mandatory. With it, the robot keeps the same `100.x.y.z` address regardless of which network it's on, and your Connection panel URL never has to change.

---

9. Troubleshooting

"Connection failed" or red dot in the Connection panel

- From the operator laptop, `curl http://<url>/feedback` — you should get JSON, even if empty (`{}`). If you don't, the network path is broken, not the bridge.
- If `curl` works but the browser doesn't, open DevTools → Network. A blocked request with no response usually means mixed content; a CORS error message means the reverse proxy isn't emitting the right headers; a `net::ERR_FAILED` at preflight stage is typically PNA (see §3).
- Confirm the bridge is actually bound to a reachable interface: on the robot host, `ss -tlnp | grep 5000` (Linux) or `Get-NetTCPConnection -LocalPort 5000` (Windows).

Chat works but tools never run / hang forever

This is the classic "forgot the tool-callback port" symptom. The UI can reach the bridge to send commands and read context, but the AI's tool calls POST to a `toolCallbackUrl` the browser can't reach.

- Inspect the agent profile in the UI — what URL is set for `toolCallbackUrl`?
- Paste that URL into the browser's address bar. You should get a 404 from a GET (the endpoint only accepts POST), *not* a connection error. A connection error means the browser can't reach the tool server, not that the URL is wrong shape.
- If the URL is `http://127.0.0.1:…` but the agent is on a different machine, you need either Option A (SSH forward of the tool port) or you need to change the agent so it advertises the public/tunnel/LAN URL.

Mixed-content / "blocked: insecure content"

The hosted UI is HTTPS. You pointed it at a plain `http://` URL that isn't loopback. Use one of: SSH tunnel (Option A), self-host the UI over HTTP (Option B), or put HTTPS in front of the agent (Options C / E / F / G).

Self-signed cert / `NET::ERR_CERT_AUTHORITY_INVALID`

You went with Option C and haven't trusted the Caddy root CA on the operator laptop yet. Either import the root cert (PowerShell snippet in §4.3) or run Tailscale + `tailscale cert` for a real Let's-Encrypt-issued cert.

CORS error in DevTools even though the bridge says `Allow-Origin: *`

- You're behind a reverse proxy that strips CORS headers. Add them at the proxy (see the Caddy snippet in §4.3).

- Chrome PNA: the proxy needs to emit `Access-Control-Allow-Private-Network: true` on the preflight response, otherwise public-origin → private-IP fetches are blocked even with HTTPS.

Port already in use on startup

```bash
# Linux / macOS
sudo lsof -iTCP:5000 -sTCP:LISTEN

# Windows
Get-Process -Id (Get-NetTCPConnection -LocalPort 5000).OwningProcess
```

Usually a previous agent run that didn't shut down cleanly. Kill the stray process, or change the port via the `OmniLinkHTTPBridge(…, port=…)` argument and update the Connection panel.

Multiple agent profiles confuse the UI

Running two ToolRunner-style agents with the same `agent_name` creates duplicate profiles that race to overwrite each other's `toolCallbackUrl`. Make sure only one instance per profile name is running, and delete stale duplicates from the Dashboard.

Tunnel URL rotates after every restart

- Cloudflare: switch from `cloudflared tunnel --url` (ephemeral) to a named tunnel + DNS record (stable).
- ngrok: paid tier gives reserved domains; otherwise pin the URL via `ngrok config` — or use Tailscale instead.

Robot stops responding mid-session

- Check the agent process is still running — `POST /command` returns 200 even if the

engine handler later fails. The `/feedback` endpoint surfaces handler errors after the fact.

- For tunnel setups, free tunnel daemons can drop connections under load. Check the `cloudflared` / `ngrok` logs.
- If the hardware bridge died (Webots crash, ROS node restart, USB disconnect), the agent process may still be up but unable to actuate. Restart the bridge and re-test with a known-safe command.

---

10. Quick Reference

Endpoints

Surface	Method & path	Purpose
Bridge	`POST /command`	UI → agent. Send a command for the engine to handle.
Bridge	`POST /inline-code`	UI → agent. Deliver an inline code snippet from the model.
Bridge	`GET /context`	UI polls the latest context the agent has published.
Bridge	`GET /feedback`	UI polls the latest handler feedback (success/error).
Tool server	`POST /tool`	UI → agent. Run a single tool call from the AI.
Tool server	`GET /activity`	UI inspects the last ~100 tool calls.

Connection panel fields

Field	Typical value
Base URL	`https://robot.your-tailnet.ts.net` · `https://abc123.trycloudflare.com` · `http://localhost:5000`
Command path	`command`
Feedback path	`feedback`
Context path	`context`
Inline-code path	`inline-code`

Field	Typical value
Username / password	Set when basic auth sits in front of the bridge.

Environment variables for the agent

Variable	Purpose
`OMNI_KEY`	Omni Key for the agent's integration user (always required).
`AXIS_PORT`	Fixed tool-callback port (default `51516`). Set this if you tunnel that port.
`AXIS_BRIDGE_URL`	OmniSim hardware bridge URL (default `http://127.0.0.1:8765`).
`AXIS_TOOL_CALLBACK_URL`	Override the tool-callback URL the agent advertises in its profile.

The one rule to remember

> The agent's URL, as the browser sees it, must be `localhost` / `127.0.0.1` / `[::1]` or an `https://` URL with a trusted certificate. Pick whichever option from §4 or §5 puts you in one of those buckets with the least friction for your team.

Next Steps

- Agent Connection — Reference for the HTTP bridge endpoints and Dashboard configuration.
- Security — Platform-wide security checklist.
- Robot Demo Guide — End-to-end example that combines a ToolRunner, the bridge, and a Pygame simulation.